Lecture 9

2 layer network

sol 2 - NEW
sol 1 - pmw
OH - 9:30, 10:30
[HW4 - TB posted

Lecture Notes IV – Neural Networks, Part 1

Marina Meilă
mmp@uwaterloo.ca

With Thanks to Pascal Poupart & Gautam Kamath
Cheriton School of Computer Science
University of Waterloo

February 2, 2026

A little history ✓

The single "neuron" ✓

Two-layer Neural Networks ←
- Hidden layer options
- Output layer options

Multi-layer neural networks ←

Training a neural network by backpropagation

Reading HTF Ch.: 11.3 Neural networks, Murphy Ch.: (16.5 neural nets), Bach Ch.: –, Deep Learning Book (Goodfellow, Bengio, Courville) 6.1-4, ResNet 7.6, ConvNet 9., Autoencoders 14.1, Dive Into Deep Learning 4.1-4.3.

(Artificial) Neural Network (nn) unit

- For each **unit** i

$$y_i \equiv f_i(x) = \phi(\sum_j w_{ij} x_j + w_{i0}) \quad (1)$$

- **Weigth vector** w_i
  - w_{ij} = strength of the link from unit i to input j
  - $w_{ij} = 0$: no link
  - w_{ij} can be positive or negative
  - Sometimes we call the input vector $x = [x_{1:d}]$ input units
- **activation function** $\phi()$
  - must be non-linear (otherwise the unit is a linear transformation)
  - wanted: monotonically increasing, differentiable, gradient non-zero [1]

non-linear, increasing

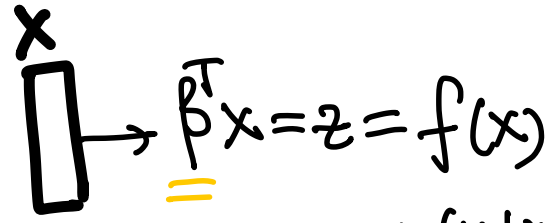
Notation ϕ is overloaded

- When we talk about nn in general: ϕ is any activation function
- When we do calculations with nn: ϕ is by default the logistic function (unless specified otherwise)

logistic or sigmoid function $\phi(u) = \frac{1}{1+e^{-u}}$ (2)

- When we do statistics or ML (but not nn): ϕ is the logistic function
- Exercise: compare $f_i(x)$ from (1) with $p(x) = Pr[y = 1 \mid x]$ from logistic regression.

[1] More technically: ϕ can be any continuous, bounded and strictly increasing function on $\mathbb{R}$.

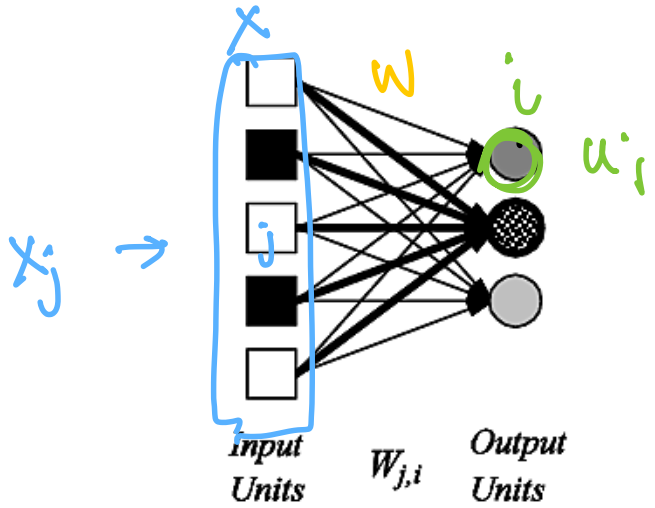


$P(y|x,\beta) = \varphi(yz)$

$= \frac{1}{1+e^{-yz}}$

Perceptron

- Single layer feed-forward network

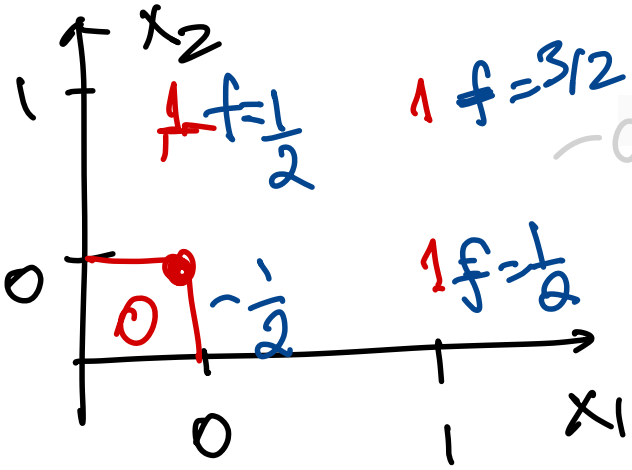


$$\varphi(z) = \frac{1}{1+e^{-z}}$$

logistic sigmoid

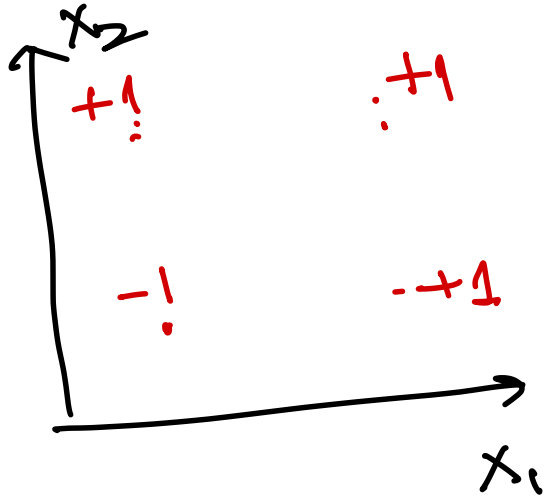
Perceptron weights for OR, AND, XOR

Exe



domain of f

OR as classification



$f^{true}(x_1, x_2) = x_1 \text{ OR } x_2$

$x_{1,2} \in \{0, 1\}$

try logistic regression

confidence $P(y|x) = \varphi(y|z)$

$w_{1,2} = 1$

$w_0 = -1/2$

(~~linear output~~) ↑ logistic

φ = activation

$f(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_0$

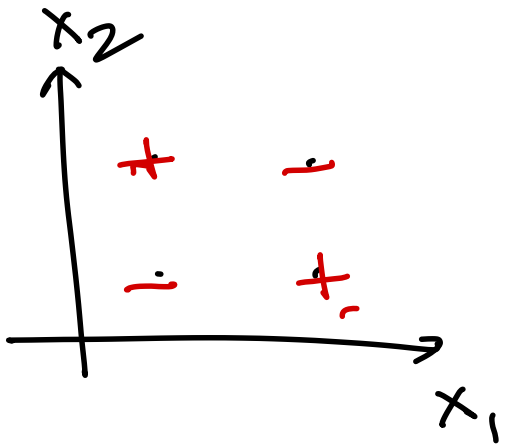
$w_0 = -1/2$

$w_1 = 1$

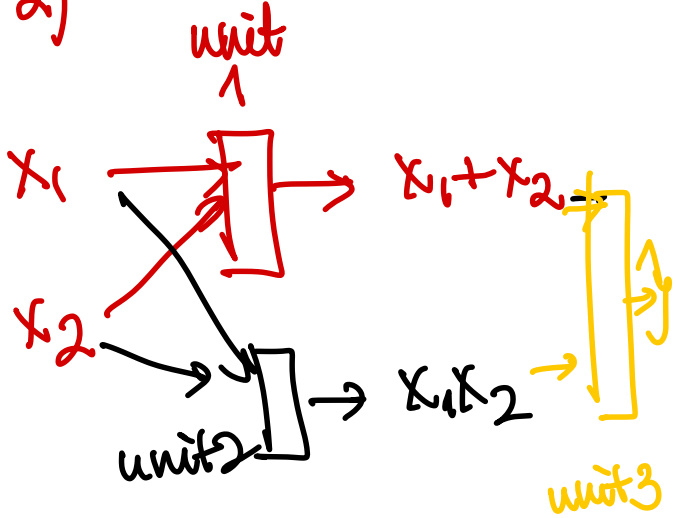
$w_2 = 1$

$\hat{y} = \frac{\operatorname{sgn} f(x_1, x_2) + 1}{2}$

Perceptron weights for OR, AND, XOR



2)



layer 1

$$f^{true}(x_1, x_2) = \frac{x_1 \text{ XOR } x_2 + 1}{2}$$

$w_0, w_1, w_2 = ?$ $\exists\ w_{0,1,2}$ so that $\text{sgn}(w_0 + w_1 x_1^i + w_2 x_2^i) = y^i$ for all i

get around

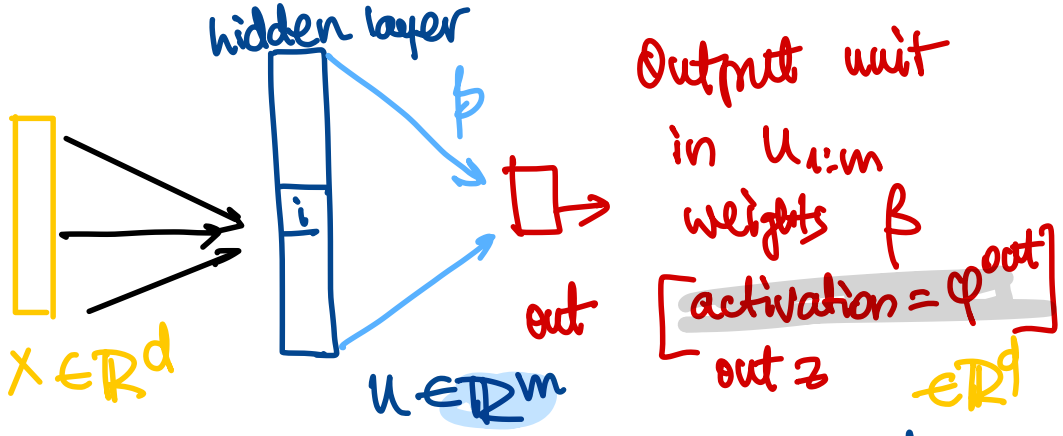
1) $\tilde{x} = [x_1\ x_2\ x_1x_2\ 1]$ ← intercept as slope

$\tilde{w} = [w_1\ w_2\ w_3\ w_0]$

$w_1 = w_2 = 1$ $\quad w_0 = -\frac{3}{4}$

$w_3 = -\frac{3}{2}$

Two-layer Neural Networks



- We build a **two-layer neural network** in the following way:

Inputs	x_j	$j = 1 : d$
Bottom layer[2]	$u_i = \phi(w_i^T x)$	$i = 1 : m$, $w_i \in \mathbb{R}^d$
Top layer	$f = \beta^T u$	$\beta \in \mathbb{R}^m$
Output	f	$\in \mathbb{R}$

unit i: input x, weights w_i, activation φ, out u_i

In other words, the neural network implements the function

$$f(x) = \sum_{i=1}^{m} \beta_i u_i = \sum_{i=1}^{m} \beta_i \phi(\sum_{j=1}^{d} w_{ij} x_j) \in (-\infty, \infty) \quad (3)$$

later — linear — linear — $z = \beta^T u$

Note that this is just a linear combination of logistic functions.

- As we will see shortly, in general, $f(x)$ can also be non-linear

[2] In neural net terminology, each variable u_i is a **unit**, the bottom layer is **hidden**, while top one is **visible**, and the units in this layer are called hidden/visible units as well. Sometimes the inputs are called **input units**; imagine neurons or individual circuits in place of each x, u, y variable.

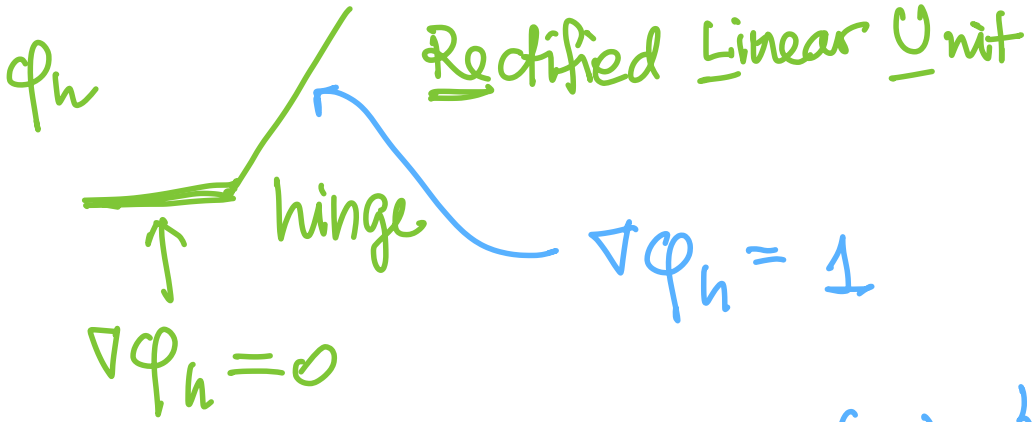
Activation functions for the hidden layer

For the hidden layer, we have to choose

- number of units m
- activation function

Common activation functions

- **Functions that approximate a step function**
  - threshold function (or step function) 1 for $u \geq 0$, and 0 otherwise (not used)
  - logistic ϕ
  - hyperbolic tangent *tanh*, arctangent $\tan^{-1}$
- **Hinge** functions
  - RELU $= max(u, 0)$
  - softplus $= \ln(1 + e^u)$
    in practical implementations, these unbounded functions are bounded at a large value M
  - Why hinge functions? Gradient is 1 or 0 (approximately), faster computation!!, and no saturation



$$\nabla\varphi_h = \begin{cases} 1, & u > 0 \\ 0, & u \leq 0 \end{cases}$$

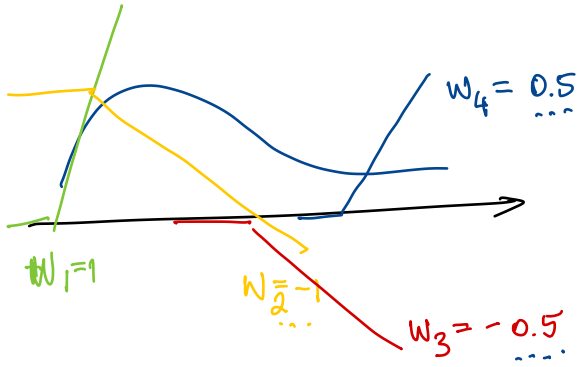
layer

$$u_1 \to \varphi_h(u_1) = \begin{cases} u_1 \\ 0 \end{cases} \quad w_1$$

$$u_2 \to \varphi_h(u_2) = \begin{cases} u_2 \\ 0 \end{cases} \quad w_2$$

$$\vdots \to \quad \vdots$$

$$\sum w_i = z = \begin{cases} u_i \neq 0 \\ w^T u \end{cases}$$


$w_4 = 0.5$
$w_1 = 1$
$w_2 = -1$
$w_3 = -0.5$

Output layer

- Let $\boxed{z = \beta^T u}$ be the linear output of the nn.
- For problems other than regression, it unifies the analysis to apply a non-linear function ϕ_{out} to z. This is why, theoretically but not in practice, we will write $f(z(x)) = \phi_{\text{out}}(z(x))$
- Why? $\phi_{\text{out}}(z)$ matches the loss $\mathcal{L}$, which in turn matches the prediction problem

Prediction problem	Predict	Output layer ϕ_{out}	Range of f
regression	$\hat{y} = z$	linear $\phi_{\text{out}} = z$	$\in \mathbb{R}$
binary classification	$\hat{y} = \text{sgn} z$	logistic $\phi_{\text{out}} = \phi(\beta^T u)$	$\in [0,1]$
multiway classification	$\hat{y} = \underset{1:r}{\text{argmax}}\, z_k$	softmax $\phi_{\text{out}\,k} = \phi_k(\beta_k^T u)$	$\in [0,1]^r$

$z_k = \beta_k^T u \quad k = 1:r$

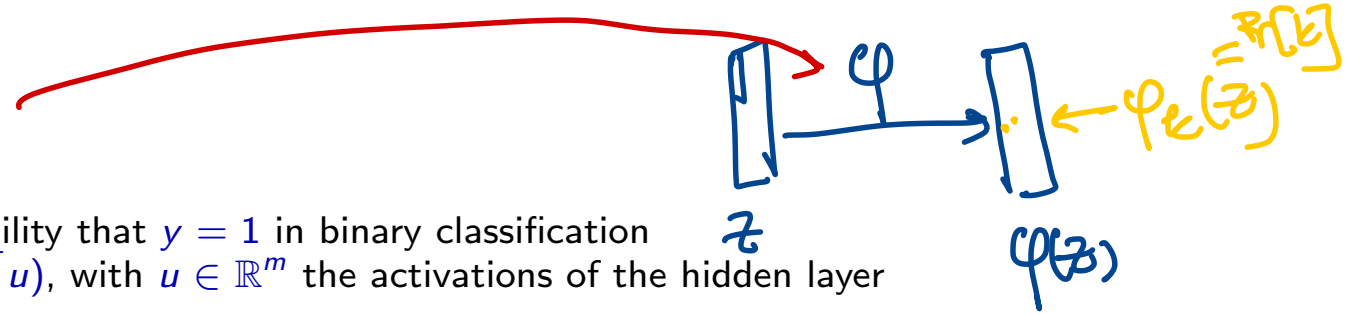
- Regression: **linear** layer as in (3) $f = \sum_i \beta_i u_i$
- Classification (binary): **logistic** layer $f(x) \in [0,1]$ is interpreted as the probability of the $+$ class.

$$f(x) = \phi\left(\sum_{j=1}^{m} \beta_i u_i\right) = \phi\left(\sum_{i=1}^{m} \beta_j \phi(\sum_j w_{ij} x_j)\right) = \Pr[y=1|x] \quad (4)$$

- Multiway classification with r classes
  - Output is vector of r functions $f_1, \ldots f_r$
  - f_k is the probability of $y = k$
  - (sometimes f_k can be a "confidence")
  - This is done with a **softmax** layer (next page)

$\varphi_k = \Pr[y=k|x]$

The softmax function



- logistic $\phi(z) = \frac{e^z}{1+e^z}$
  - represents the probability that $y = 1$ in binary classification
  - in a nn, $f(x) = \phi(\beta^T u)$, with $u \in \mathbb{R}^m$ the activations of the hidden layer
- The softmax function generalizes the logistic to r classes $\phi(z) : \mathbb{R}^r \rightarrow (0,1)^r$

$$\phi_k(z) = \frac{e^{z_k}}{\sum_{j=1}^m e^{z_j}}, \text{ for } k = 1 : r \quad (5)$$

$$\phi(z) = [\phi_1(z) \ldots \phi_r(z)] \text{ (overload of } \phi) \quad (6)$$

- Properties of softmax
  - $\sum_{j=1}^m \phi_j(z) = 1$ for all z
  - for $z_k \gg z_j$, $\phi_k(z) \rightarrow 1$.
  - derivatives $\frac{\partial \phi_j}{\partial z_k} = \phi_k \delta_{jk} - \phi_j \phi_k$
- in a nn, with output activations $u \in \mathbb{R}^m$, we train r weight vectors $\beta_{1:r}$, and

$$f_k(x) = \phi_k(\beta_k^T u(x)), \quad \text{for } k = 1 : r \quad (7)$$

Are multiple layers necessary?

- **1990's: NO**
- **2000's: YES**
- **2020's: The more the better!**

- A theoretical result

Theorem (Cybenko,≈1986)

Any continuous function from $[0,1]^d$ *to* $\mathbb{R}$ *can be approximated arbitrarily closely by a linear output, two layer neural network defined in* (3) *with a sufficiently large number of hidden units* m.

- A practical result

Deep learning = multi-layer neural net

- So, what is new?
  - small variations in the "units", e.g. switch stochastically w.p. $\phi(w^T x^{in})$ (Restricted Bolzmann Machine), Rectified Linear units
  - training method stochastic gradient, auto-encoders vs. back-propagation (we will return to this when we talk about training predictors)
  - lots of data
  - **double descent**